Kauno technologijos universitetas

Informatikos fakultetas

BST ir AVL medžiai

P175B014 Duomenų struktūrų antras laboratorinis darbas

Projekto autorius

Gustas Klevinskas

Akademinė grupė

IFF-8/7

Kaunas, 2019

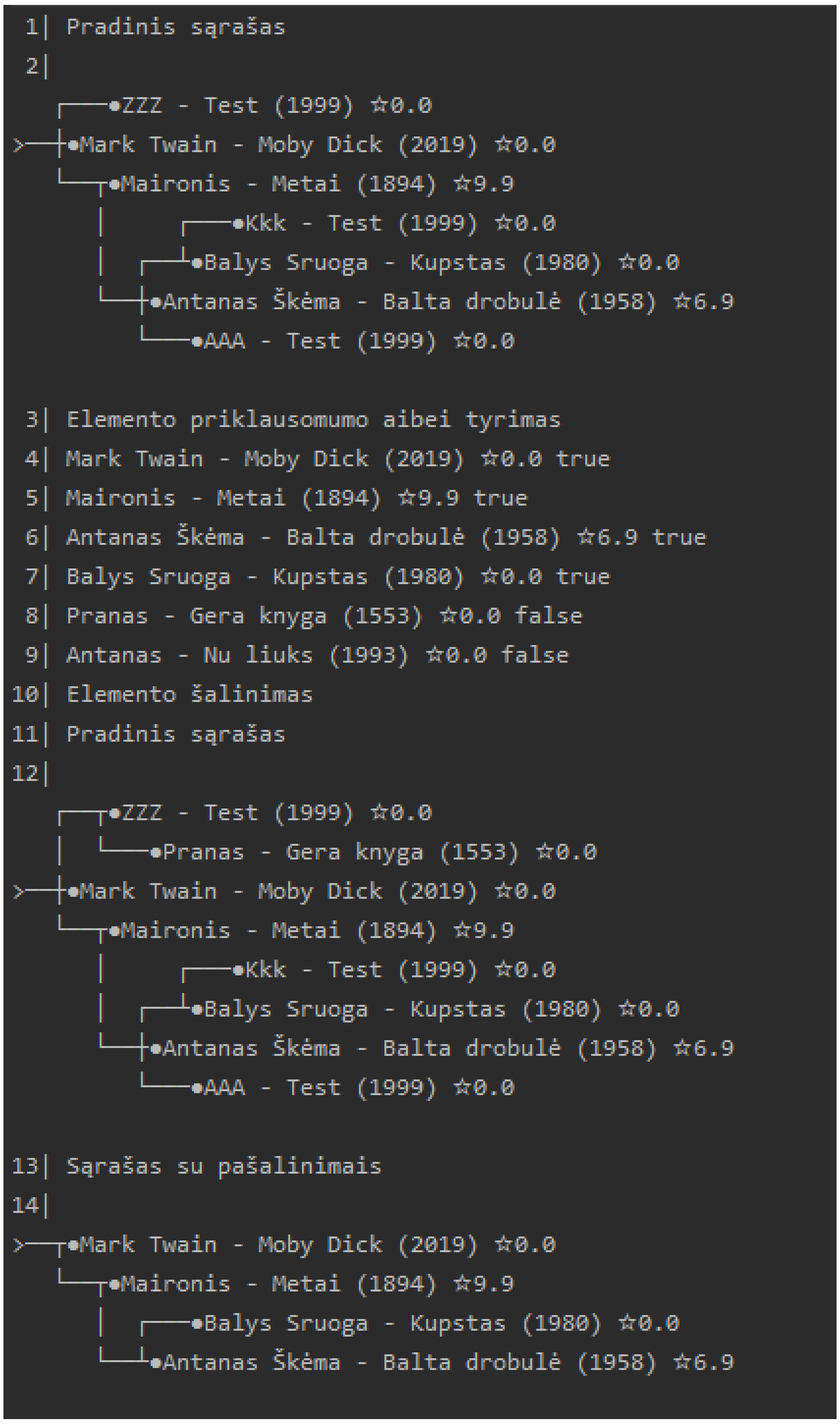
Turinys

Savo sukurtos klasės testavimas

Kodas:

```
private static void generateBooks() {
    Book b1 = new Book("Mark Twain", "Moby Dick", 2019);
    Book b2 = new Book("Maironis", "Metai", 1894);
    Book b3 = new Book("Antanas Škėma", "Balta drobulė", 1958);
    Book b4 = new Book("Balys Sruoga", "Kupstas", 1980);

    b2.setRating(9.9f);
    b3.setRating(6.9f);

    books = new Book[]{b1, b2, b3, b4};

    booksBstSet.add(b1);
    booksBstSet.add(b2);
    booksBstSet.add(b3);
    booksBstSet.add(b4);
    booksBstSet.add(new Book("ZZZ", "Test", 1999));
    booksBstSet.add(new Book("AAA", "Test", 1999));
    booksBstSet.add(new Book("Kkk", "Test", 1999));
}

private static void executeTest() {
    generateBooks();

    // Initial set
    Ks.oun("Pradinis sąrašas");
    Ks.oun("\n" + booksBstSet.toVisualizedString(""));

    // Check if contains
    Ks.oun("Elemento priklausomumo aibei tyrimas");
    for (Book i : books)
        Ks.oun(i + " " + booksBstSet.contains(i));
    Book b1 = new Book("Pranas", "Gera knyga", 1553);
    Book b2 = new Book("Antanas", "Nu liuks", 1993);
    Ks.oun(b1 + " " + booksBstSet.contains(b1));
    Ks.oun(b2 + " " + booksBstSet.contains(b2));
    booksBstSet.add(b1);

    // Delete from set
    Ks.oun("Elemento šalinimas");
    Ks.oun("Pradinis sąrašas");
    Ks.oun("\n" + booksBstSet.toVisualizedString(""));
    booksBstSet.remove(b1);
    booksBstSet.remove(new Book("Maironis", "Metai", 1894));
    booksBstSet.remove(new Book("ZZZ", "Test", 1999));
    booksBstSet.remove(new Book("AAA", "Test", 1999));
    booksBstSet.remove(new Book("Kkk", "Test", 1999));
    Ks.oun("Sąrašas su pašalinimais");
    Ks.oun("\n" + booksBstSet.toVisualizedString(""));
}
```

Rezultatai:



```
 1| Pradinis sąrašas
 2|
   ┌──•ZZZ - Test (1999) ☆0.0
>──┼•Mark Twain - Moby Dick (2019) ☆0.0
   └──┬•Maironis - Metai (1894) ☆9.9
      │     ┌──•Kkk - Test (1999) ☆0.0
      │  ┌──┴•Balys Sruoga - Kupstas (1980) ☆0.0
      └──┼•Antanas Škėma - Balta drobulė (1958) ☆6.9
         └──•AAA - Test (1999) ☆0.0

 3| Elemento priklausomumo aibei tyrimas
 4| Mark Twain - Moby Dick (2019) ☆0.0 true
 5| Maironis - Metai (1894) ☆9.9 true
 6| Antanas Škėma - Balta drobulė (1958) ☆6.9 true
 7| Balys Sruoga - Kupstas (1980) ☆0.0 true
 8| Pranas - Gera knyga (1553) ☆0.0 false
 9| Antanas - Nu liuks (1993) ☆0.0 false
10| Elemento šalinimas
11| Pradinis sąrašas
12|
   ┌──┬•ZZZ - Test (1999) ☆0.0
   │  └──•Pranas - Gera knyga (1553) ☆0.0
>──┼•Mark Twain - Moby Dick (2019) ☆0.0
   └──┬•Maironis - Metai (1894) ☆9.9
      │     ┌──•Kkk - Test (1999) ☆0.0
      │  ┌──┴•Balys Sruoga - Kupstas (1980) ☆0.0
      └──┼•Antanas Škėma - Balta drobulė (1958) ☆6.9
         └──•AAA - Test (1999) ☆0.0

13| Sąrašas su pašalinimais
14|
>──┬•Mark Twain - Moby Dick (2019) ☆0.0
   └──┬•Maironis - Metai (1894) ☆9.9
      │  ┌──•Balys Sruoga - Kupstas (1980) ☆0.0
      └──┴•Antanas Škėma - Balta drobulė (1958) ☆6.9
```

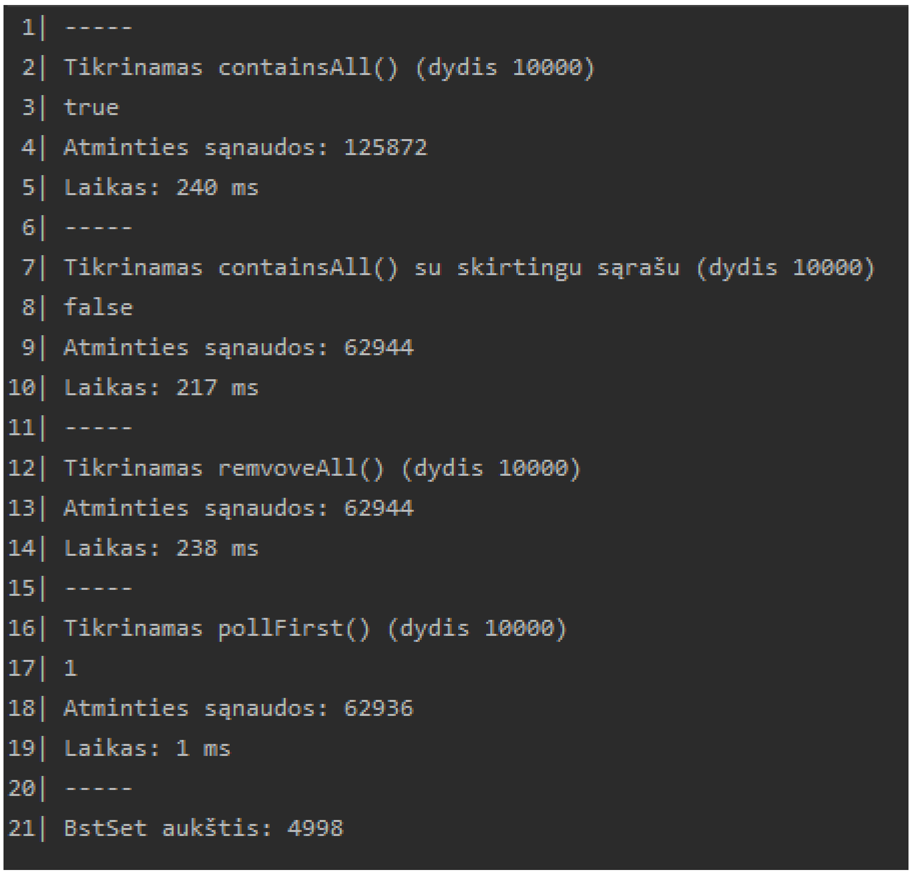
BST metodai

Kodas:

```
public boolean containsAll(BstSet<?> c) {
    Iterator i = c.iterator();

    while (i.hasNext()) {
        if (!contains((E) i.next()))
            return false;
```

```java
    }

    return true;
}

public void removeAll(BstSet<?> c) {
    Iterator i = c.iterator();

    while (i.hasNext()) {
        remove((E) i.next());
    }
}

public E pollFirst() {
    if (size == 0)
        return null;

    BstNode<E> node = getMin(root);
    remove(node.element);
    return node.element;
}
```

Rezultatai:

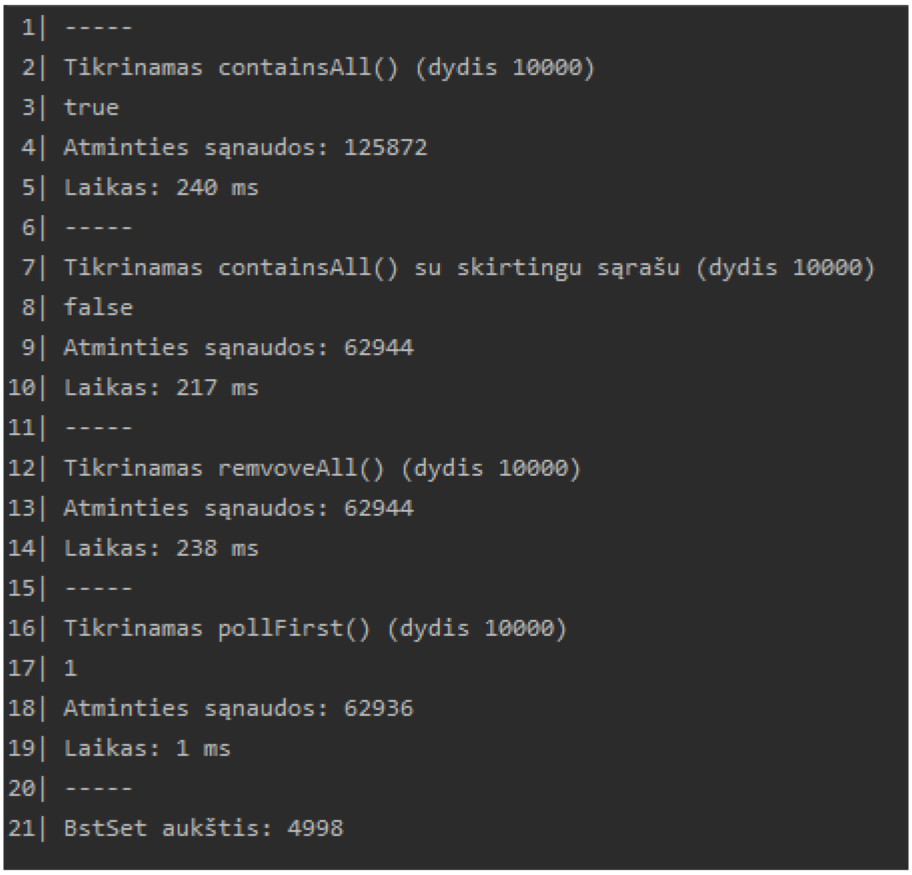


```
-----
Tikrinamas containsAll() (dydis 10000)
true
Atminties sąnaudos: 125872
Laikas: 240 ms
-----
Tikrinamas containsAll() su skirtingu sąrašu (dydis 10000)
false
Atminties sąnaudos: 62944
Laikas: 217 ms
-----
Tikrinamas remvoveAll() (dydis 10000)
Atminties sąnaudos: 62944
Laikas: 238 ms
-----
Tikrinamas pollFirst() (dydis 10000)
1
Atminties sąnaudos: 62936
Laikas: 1 ms
-----
BstSet aukštis: 4998
```

Medžio aukštis

Kodas:

```java
public int getHeight() {
    return getHeightRecursive(root);
}

private int getHeightRecursive(BstNode<E> node) {
    if (node == null)
        return -1;

    int leftHeight = getHeightRecursive(node.left);
    int rightHeight = getHeightRecursive(node.right);

    if (leftHeight > rightHeight)
        return leftHeight + 1;
    else
        return rightHeight + 1;
}
```
Rezultatai:



```
-----
Tikrinamas containsAll() (dydis 10000)
true
Atminties sąnaudos: 125872
Laikas: 240 ms
-----
Tikrinamas containsAll() su skirtingu sąrašu (dydis 10000)
false
Atminties sąnaudos: 62944
Laikas: 217 ms
-----
Tikrinamas remvoveAll() (dydis 10000)
Atminties sąnaudos: 62944
Laikas: 238 ms
-----
Tikrinamas pollFirst() (dydis 10000)
1
Atminties sąnaudos: 62936
Laikas: 1 ms
-----
BstSet aukštis: 4998
```

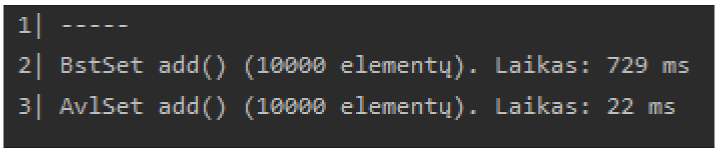
BstSet ir AvlSet greitaveika

add()

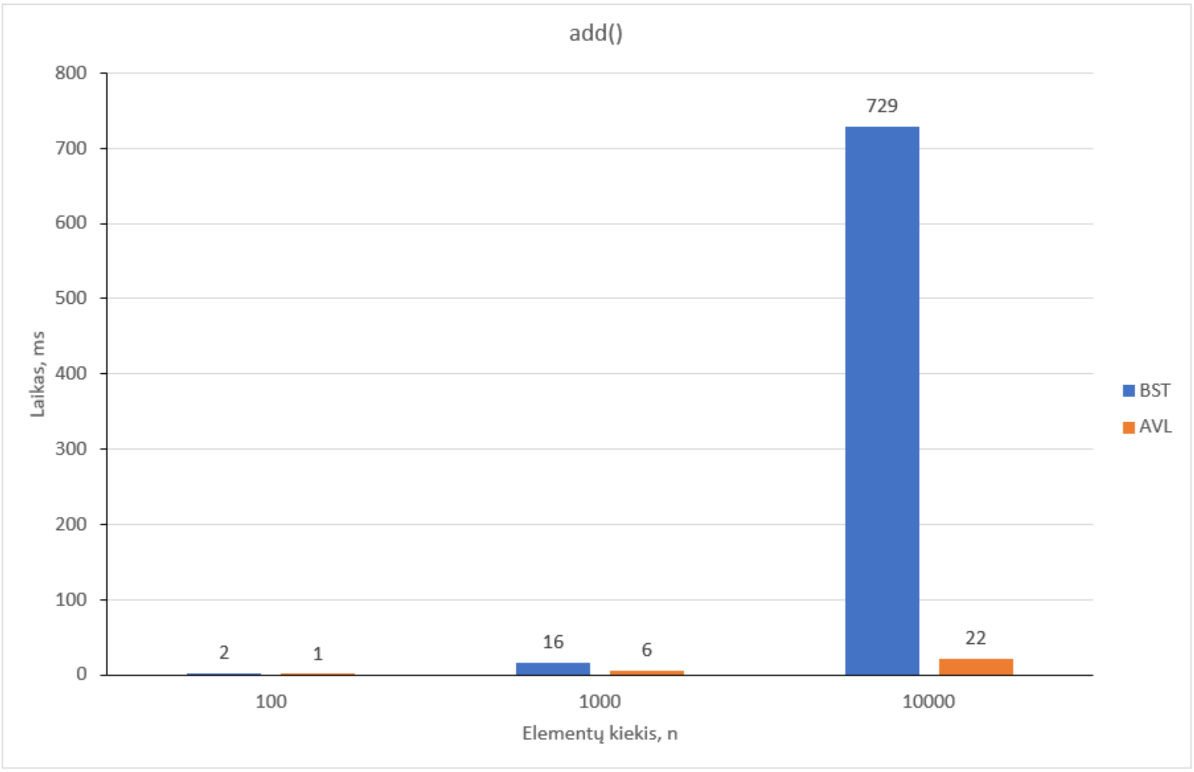
Kodas:

```java
private static void bstAvlAdd(int n) {
    BstSet<Integer> bstSet = new BstSet<>();
    AvlSet<Integer> avlSet = new AvlSet<>();

    long startTime = System.currentTimeMillis();
    for (int i = 0; i < n; i++)
        bstSet.add(i);
    long endTime = System.currentTimeMillis();
    bstTime = endTime - startTime;
    Ks.oun("-----");
    Ks.oun("BstSet add() (" + n + " elementų). Laikas: " + bstTime + " ms");

    startTime = System.currentTimeMillis();
    for (int i = 0; i < n; i++)
        avlSet.add(i);
    endTime = System.currentTimeMillis();
    avlTime = endTime - startTime;
    Ks.oun("AvlSet add() (" + n + " elementų). Laikas: " + avlTime + " ms");
}
```

Rezultatai:



```
-----
BstSet add() (10000 elementų). Laikas: 729 ms
AvlSet add() (10000 elementų). Laikas: 22 ms
```



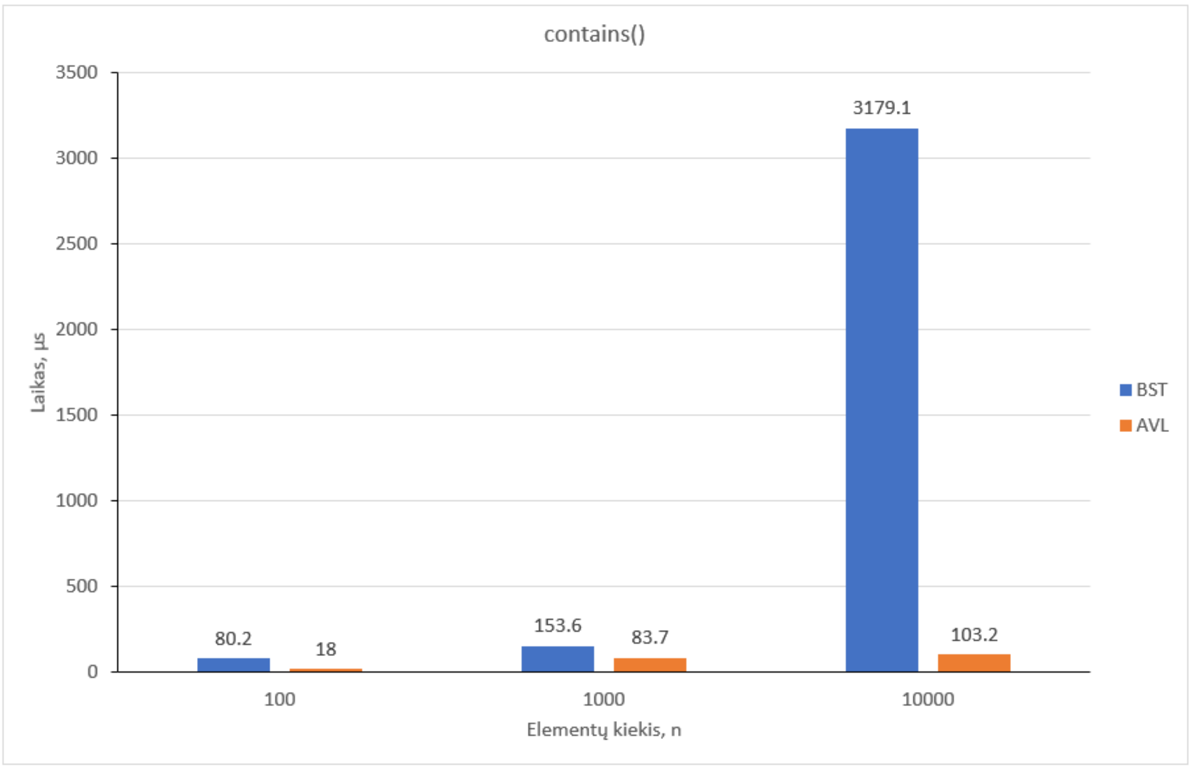
contains()

Kodas:

```
private static void bstAvlContains(int n) {
    BstSet<Integer> bstSet = new BstSet<>();
    AvlSet<Integer> avlSet = new AvlSet<>();

    for (int i = 0; i < n; i++) {
        bstSet.add(i);
        avlSet.add(i);
    }
    int randomInt = new Random().nextInt(n);
    long startTime = System.nanoTime();
    bstSet.contains(randomInt);
    long endTime = System.nanoTime();
    bstTime = endTime - startTime;
    Ks.oun("-----");
    Ks.oun("Ieškoma skaičiaus " + randomInt);
    Ks.oun("BstSet contains() (" + n + " elementų). Laikas: " + bstTime + " ns");

    startTime = System.nanoTime();
    avlSet.contains(randomInt);
    endTime = System.nanoTime();
    avlTime = endTime - startTime;
    Ks.oun("AvlSet contains() (" + n + " elementų). Laikas: " + avlTime + " ns");
}
```

Rezultatai:

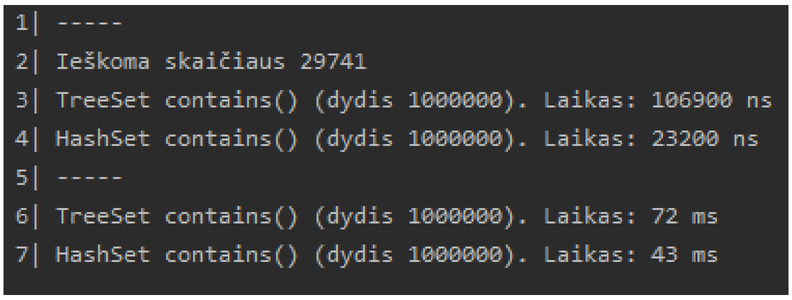
```
-----
Ieškoma skaičiaus 5433
BstSet contains() (10000 elementų). Laikas: 529100 ns
AvlSet contains() (10000 elementų). Laikas: 93800 ns
```



TreeSet ir HashSet greitaveika

Kodas:

```
private static void greitaveika2(int n) {
    TreeSet<Integer> treeSet = new TreeSet<>();
    HashSet<Integer> hashSet = new HashSet<>();
    long startTime;
    long endTime;

    for (int i = 0; i < n; i++) {
        treeSet.add(i);
        hashSet.add(i);
    }

    int randomInt = new Random().nextInt(n);
    Ks.oun("-----");
    Ks.oun("Ieškoma skaičiaus " + randomInt);

    startTime = System.nanoTime();
    treeSet.contains(randomInt);
    endTime = System.nanoTime();
    Ks.oun("TreeSet contains() (dydis " + n + "). Laikas: " + (endTime -
startTime) + " ns");

    startTime = System.nanoTime();
    treeSet.contains(randomInt);
    endTime = System.nanoTime();
    Ks.oun("HashSet contains() (dydis " + n + "). Laikas: " + (endTime -
startTime) + " ns");
}

private static void greitaveika3(int n) {
    TreeSet<Integer> treeSet = new TreeSet<>();
    HashSet<Integer> hashSet = new HashSet<>();
    Collection<Integer> collection = new ArrayList<>();
    long startTime;
    long endTime;

    for (int i = 0; i < n; i++) {
        treeSet.add(i);
        hashSet.add(i);
        if (i % 5 == 0)
            collection.add(i);
    }

    Ks.oun("-----");

    startTime = System.currentTimeMillis();
    treeSet.containsAll(collection);
    endTime = System.currentTimeMillis();
    Ks.oun("TreeSet contains() (dydis " + n + "). Laikas: " + (endTime -
startTime) + " ms");

    startTime = System.currentTimeMillis();
    treeSet.containsAll(collection);
    endTime = System.currentTimeMillis();
    Ks.oun("HashSet contains() (dydis " + n + "). Laikas: " + (endTime -
startTime) + " ms");
}
```
Rezultatai:



```
-----
Ieškoma skaičiaus 29741
TreeSet contains() (dydis 1000000). Laikas: 106900 ns
HashSet contains() (dydis 1000000). Laikas: 23200 ns
-----
TreeSet contains() (dydis 1000000). Laikas: 72 ms
HashSet contains() (dydis 1000000). Laikas: 43 ms
```

BstSet metodai

headSet()

```
public Set<E> headSet(E element) {
    if (element == null) {
        throw new IllegalArgumentException("Element is null in headSet(E
element)");
    }

    Set<E> newSet = new BstSet<>();
    headSetRecursive(newSet, element, root);
    return newSet;
}

private void headSetRecursive(Set<E> newList, E element, BstNode<E> node) {
    if (node != null) {
        if (c.compare(element, node.element) > 0)
            newList.add(node.element);

        headSetRecursive(newList, element, node.left);
        headSetRecursive(newList, element, node.right);
    }
}
```

subSet()

```
public Set<E> subSet(E from, E to) {
    if (from == null || to == null) {
        throw new IllegalArgumentException("Element is null in subSet(E from, E
to)");
    }

    Set<E> newSet = new BstSet<>();
    subSetRecursive(newSet, from, to, root);
    return newSet;
}

private void subSetRecursive(Set<E> newList, E from, E to, BstNode<E> node) {
    if (node != null) {
        if ((c.compare(from, node.element) <= 0) && (c.compare(to, node.element) >
0))
            newList.add(node.element);

        subSetRecursive(newList, from, to, node.left);
        subSetRecursive(newList, from, to, node.right);
    }
}
```

tailSet()

```
public Set<E> tailSet(E element) {
    if (element == null) {
        throw new IllegalArgumentException("Element is null in tailSet(E
element)");
    }

    Set<E> newSet = new BstSet<>();
    tailSetRecursive(newSet, element, root);
    return newSet;
}

private void tailSetRecursive(Set<E> newList, E element, BstNode<E> node) {
    if (node != null) {
        if (c.compare(element, node.element) <= 0)
            newList.add(node.element);

        tailSetRecursive(newList, element, node.left);
        tailSetRecursive(newList, element, node.right);
    }
}
```

Iteratoriaus remove()

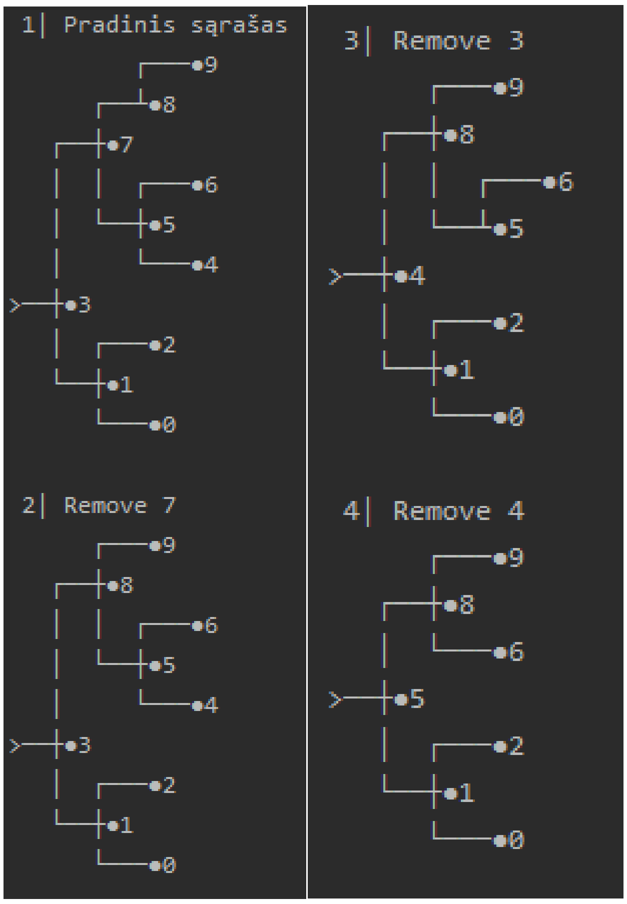
```
public void remove() {
    if (!stack.empty()) {
        BstNode<E> n = stack.pop();
        parent = (!stack.empty()) ? stack.peek() : root;
        BstNode<E> node = (ascending) ? n.right : n.left;
        toStack(node);

        parent = removeRecursive(n.element, parent);
    }
}
```

AvlSet remove()

Kodas:

```
public void remove(E element) {
    root = removeRecursive(element, (AVLNode<E>) root);
}

private AVLNode<E> removeRecursive(E element, AVLNode<E> n) {
    if (n == null) {
        return null;
    }

    int cmp = c.compare(element, n.element);

    if (cmp < 0) {
        n.left = removeRecursive(element, n.getLeft());
    } else if (cmp > 0) {
        n.right = removeRecursive(element, n.getRight());
    } else {
        if ((n.getLeft() == null) || (n.getRight() == null)) {
            if (n.getLeft() == null)
                n = n.getRight();
            else
                n = n.getLeft();
        } else {
            // Node with two children: get the inorder
            // successor (smallest in the right subtree)
            AVLNode<E> temp = minValueNode(n.getRight());

            // Copy the inorder successor's data to this node
            n.element = temp.element;

            // Delete the inorder successor
            n.right = removeRecursive(temp.element, n.getRight());
        }
    }
    // If the tree had only one node then return
    if (n == null)
        return null;

    n.height = Math.max(height(n.getLeft()), height(n.getRight())) + 1;

    // Get the balance factor of this node (to check whether
    // this node became unbalanced)
    int balance = getBalance(n);

    // If this node becomes unbalanced, then there are 4 cases
    // Left Left Case
    if (balance > 1 && getBalance(n.getLeft()) >= 0)
        return rightRotation(n);

    // Left Right Case
    if (balance > 1 && getBalance(n.getLeft()) < 0) {
        n.left = leftRotation(n.getLeft());
        return rightRotation(n);
    }

    // Right Right Case
    if (balance < -1 && getBalance(n.getRight()) <= 0)
        return leftRotation(n);

    // Right Left Case
    if (balance < -1 && getBalance(n.getRight()) > 0) {
        n.right = rightRotation(n.getRight());
        return leftRotation(n);
    }

    return n;
}
```

Rezultatai:



```
 1│ Pradinis sąrašas
          ┌────•9
      ┌───┴•8
  ┌───┼•7
  │   │   ┌────•6
  │   └───┼•5
  │       └────•4
>─┼•3
  │   ┌────•2
  └───┼•1
      └────•0
```

```
 2│ Remove 7
      ┌────•9
  ┌───┼•8
  │   │   ┌────•6
  │   └───┼•5
  │       └────•4
>─┼•3
  │   ┌────•2
  └───┼•1
      └────•0
```

```
 3│ Remove 3
       ┌────•9
  ┌────┼•8
  │    │    ┌────•6
  │    └────┴•5
>─┼•4
  │    ┌────•2
  └────┼•1
       └────•0
```

```
 4│ Remove 4
       ┌────•9
  ┌────┼•8
  │    └────•6
>─┼•5
  │    ┌────•2
  └────┼•1
       └────•0
```